Fourth European Conference on Artificial Life (ECAL97), Brighton
http://www.cogs.susx.ac.uk/ecal97/
Brighton, 1997

A Topological Structure Based on Hashing - Emergence of a "Spatial" Organisation

Peter Dittrich and Wolfgang Banzhaf
University of Dortmund, Dept. of Computer Science, D-44221 Dortmund (Germany)
dittrich@icd.de, banzhaf@icd.de, http://ls11-www.informatik.uni-dortmund.de/

Abstract

A topological structure based on hashing for an algorithmic reaction system is introduced. Hashing, as a very efficient storage method for certain problems, uses an important property of the computer: Its ability to accurately access a specific address in memory space. It is shown, how a self-organising system can be built with this method. We discuss experiments with a reaction system based on the resulting hash topology.

Keywords: artificial life, self-evolution, prebiotic evolution, molecular computer, self-programming, autokatalytic reaction system, hypercycle, self-replication

1 Introduction

Computer models can be used to study complex phenomena of self-organization. Life itself has become the topic of such studies in recent years [13]. A sub-class of models are systems that are inspired by chemical reaction systems [8, 15, 18]. Other simulations intend to model other levels of life-like behavior, e.g. the evolution of cells and organisms [14, 16].

These models nearly always apply the Euclidean topology of physical space [1] if spatial structures are used. Even Tierra [16] is influenced by the euclidean space metaphor. There, the pattern search mechanism induces a one dimensional space.

In this paper we will introduce a new method for creating a "spatial" world with self-organizing topology that tries to respect the medium "computer" as much as possible[1]. The method will be based on hashing.

Our goals are primarily:

(1) *To introduce a method for creating a self-organizing topology.*
In systems like AlChemy [8] or machine-tape-rewriting systems [10, 18] the same object is interpreted in two ways. (1) It can be seen as pure, passive data, or (2) it can be transformed (folded) to an active operator (machine) which is able to process other objects as data in order to produce new objects. Here, we will introduce a third interpretation. Besides the dualism of data and operator, the representation of an object will be used to constitute a topological relation among the objects (Fig. 1).

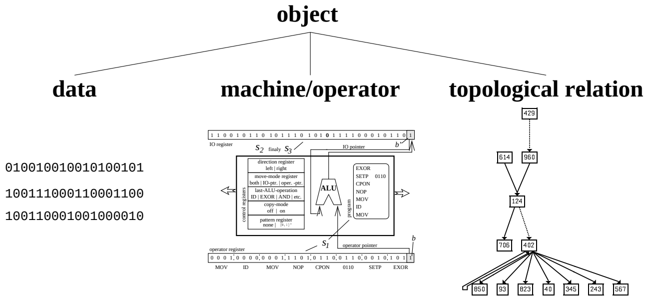

Figure 1: *Three functions of an object: (1) data, (2) machine and (3) spatial structure.*

(2) *To show that the presented method is feasible and has the desired properties.*
This is done by three series of experiments which show that introducing the hashing-topology increases the phenomenally repertoire of an autokatalytic reaction system. The experiments will not allow general conclusions about spatially self-organizing systems.

(3) *To show that information processing systems used in computer science have inherent properties for self-organisation and evolution.*
Two concepts of computer science, a finite state machine and a hashing function, are joined by a simple algorithm. The resulting dynamical system shows complex self-organizing behaviours, which can not be assigned to the surrounding algorithm but have to be seen as inherently rooted in the computational subsystems.

The motivation of this work is not to model a certain biological system. Rather, the system presented here can be seen as a model for (spatially) self-organizing systems.

[1] With the term "computer" we refer to today's conventional von-Neumann computer.

2 The Hashing Method

Hashing is an efficient and widely used technique in Computer Science for implementing dictionaries or sets. By careful design, storage management by hashing requires only a constant time per lookup. The aim of hashing is to find a certain object $s \in S$ from the set S specified by a **key** $k = key(s), k \in K$ in a large memory array.

The essential idea is that the (possibly infinite) set S of objects is partitioned into a number n_B of "classes" with finite content by the **hash function**

$$hash : K \rightarrow \{0, 1, \ldots, n_B - 1\}. \tag{1}$$

The classes are called **buckets** and each object s belongs to bucket $hash(key(s))$. In order to look up an object of the memory array, its corresponding bucket number $hash(key(s))$ is calculated and the (fixed-sized) bucket is searched for the object.

Problems arise when more objects should be stored in a bucket then it can hold: If we try to place s in a bucket $hash(key(s))$ and find it already filled up, a "collision" occurs. In this situation a rehash strategy chooses a set of alternative locations in which storage is tried. In the worst case, this rehashing requires time proportional to the number of objects stored in the memory array.

An example to illustrate the hashing method is the following table. It shows a memory array (called hash table) where the stored objects are pairs of name and phone number. The key is the name. The hash value $hash(name)$ is defined as the number characters in $name$:

bucket number	key	phone number
0	-empty-	-empty-
1	-empty-	-empty-
2	-empty-	-empty-
3	Ted	0234 4711
4	-empty-	-empty-
5	Peter	0231 9700
6	Walter	0042 8146
7	-empty-	-empty-
...	...	...
n_B	...	...

3 Generating Topology through Hashing

In contrast to the real world, distances between memory cells is independent of their position in a computer. [2]. For example, the CPU time a copy instruction consumes is constant no matter to what location the data is transferred. This is an important difference when studying life, because recent work suggests that topological structures play an important role in the evolution of life.

[2] In reality, different memory types exist and transfer between them takes different time.

In artificial life ("AL") systems relationships among systems or sub-systems are implemented by various techniques which are consuming computational resources like CPU time and memory space. For example, pattern matching is often used in Tierra-like systems for addressing memory locations. In other models the relationships are stored explicitly as a graph which may need a lot of memory.

In systems like "The Game of Life" [6] cells occupy a grid inspired by Euclidean topology. Already Ray has pointed out, however, that such an "space" does not exist in a computer, and that when we intend to instantiate an AL system in a computer we should "respect the medium" [16]. We will follow this philosophy by using the hashing method for generating a topology because accessing a memory location via a bucket number is a more natural and adapted way to manipulate memory in a computer.

In the following we shall see what memory access has to do with topology for interactions. Suppose two objects s_1 and s_2 interact and produce a result s_3. If there is no topological structure s_3 can be stored at any random position in memory. This corresponds to a well-stirred reactor or a "bag full of enzymes".

We could define a neighborhood of an object s by introducing a Euclidean topology for objects. Alternatively, we could simply use a hash function $hash(key(s))$ which would always return the bucket number s is related to. This would generate a natural topology for objects s. It should be noted that this topology is under control of the objects (molecules, individuals) themselves. There is no algorithm "on top" of the objects, which determines the connection among elements, like, e.g., in a system where an Evolutionary Algorithm evolves a Neural Network. The phenomenon that a system actively manipulates its spatial relationships can be found in nature, too. Examples are the cytosceleton or the human-made traffic/transportation system.

4 The Binary String System

This section describes the system where we have applied hashing.

The system is inspired by (bio-)chemical reaction systems like the RNA world, where molecules collide and interact to produce new molecules [3]. It can be called an **artificial chemistry**, because the interaction rules are defined by abstract algorithms. Through this abstraction one tries to extract the logic of information processing rather than simulating physical details [5].

The system consists of the following three components:

(1) *A soup (population) of objects.*
These objects may be character sequences [11], lambda-expressions [8], binary strings [18, 2] or numbers. Here, we use binary strings with a constant

length of 32 bit.

(2) *A collision or reaction rule.*
A collision rule defines the interaction among two objects s_1 and s_2 which may lead to the generation of new objects, e.g. s_3.

(3) *An algorithm to run the system.*
Special instances of the algorithm used in this contribution can also be found with minor modifications in [8, 11, 18, 2].

4.1 The soup

The soup consists of a collection of n_B buckets. Each bucket contains n_{bs} objects. The total soup size is $M = n_B \times n_{bs}$. As already said before, an object here is a binary string $s \in \{0,1\}^{32}$ of fixed size 32.

4.2 The collision rule

The collision rule is motivated by the RNA world where the molecules have two functions. They can either act as enzymes or be processed by other enzymes. In the first role they act as active machines and in the second role they are processed as passive data. This dualism can already be found in conventional computer hardware where binary strings represent programs as well as data.

We call the special reaction rule **automata reaction**, because it is based on a finite state automaton which is a blend of a Turing-machine and a register machine. The Typogenetics of Hofstadter [9] has inspired this model.

The automata reaction instantiates a deterministic reaction $s_1 + s_2 \Longrightarrow s_3$, where $s_1, s_2, s_3 \in \{0,1\}^{32}$. In order to calculate the product string s_3, string s_1 is "folded" into the instruction for an automaton A_{s_1}, which gets s_2 as an input. The construction of the automaton ensures that it will always halt after a finite number of steps. Because the automaton is finite and deterministic, this reaction rule defines a functions $\{0,1\}^{32} \times \{0,1\}^{32} \longrightarrow \{0,1\}^{32}$.

Figure 2 shows the structure of the automaton. It contains two 32-bit registers, the **IO register** and the **operator register**. At the outset, operator string s_1 is written into the operator register and operand s_2 into the IO register. The program is generated from s_1 by simply mapping successive 4-bit segments into instructions. The resulting program is executed sequentially, starting with the first instruction. There are no control statements for loops or jumps in the instruction set.

Each 32-bit register has a pointer, referring to a bit location. The **IO pointer** refers to a bit b' in the IO register and the **operator pointer** refers to a bit b in the operator register. Bits b and b' are inputs to the **ALU**. The ALU operation result is stored at the IO pointer's location replacing b'.

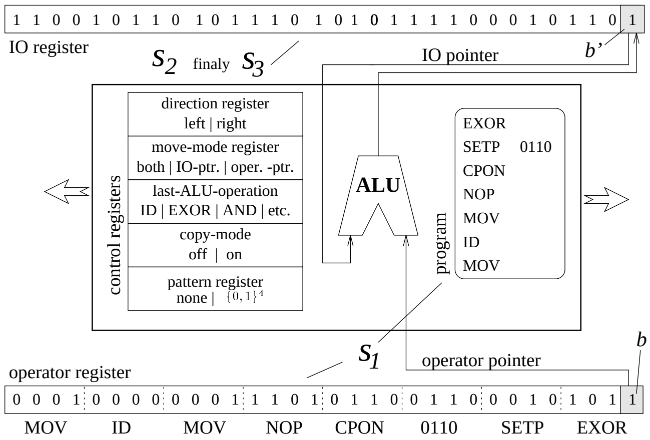

Figure 2: *The finite and deterministic automaton used. It carries out the reaction $s_1 + s_2 \Longrightarrow s_3$. s_1 is written into the operator register and specifies the program. The I/O register is initialized with s_2 and contains, after executing the program, the result string s_3.*

For a precise formal specification of the automata reaction, see the source code, available under [7].

5 The Hash-reactor Algorithm

The following general algorithm is used to run the binary string system:

$b_1 \leftarrow chooseBucket_1()$
$s_1 \leftarrow getObj(b_1)$
$b_2 \leftarrow chooseBucket_2(s_1, b_1)$
$s_2 \leftarrow getObj(b_2)$
$products \leftarrow collide(s_1, s_2)$
for each $s_3 \in products$ do
  $b_3 \leftarrow chooseBucket_3(s_1, s_2, b_1, b_2)$
  $storeObj(s_3, b_3)$
done
goto 1 (iterate)

s_1, s_2, s_3 are binary strings and $b_1, b_2, b_3 \in \{0, 1, \ldots, n_B - 1\}$. The function $chooseBucket_i$ selects a bucket number. For $chooseBucket_2$ and $chooseBucket_3$ a hash function can be used to calculate b_2 and b_3, respectively. The function $getObj(b)$ returns a randomly selected string from bucket b. The procedure $storeObj(s, b)$ replaces a string in bucket b by s. The strategy for the replacement is first-in-last-out.

M iterations of the algorithm are called a **generation**. This notion is used to generate a time scale. When using "generation" instead of "number of iteration" the comparison of runs with different soup sizes becomes more straight forward.

6 Visualisation

6.1 Macroscopic Measurements

To analyze the experiments we have used the following measurements. These methods try to observe the whole system, by mapping its high-dimensional state into a low-dimensional space which can be displayed over time.

Diversity (Div):

The diversity *Div* is simply defined as the number of different string types in the soup.

Innovativity (Inn):

The (total) innovativity at time step t for the time window $[0, t]$ is the average number of string types — generated in one iteration of the algorithm — that have not appeared before in the system during the time window.

Distance Distribution Complexity:

To measure the complexity of the mix the distance distribution complexity ("DDC") [12] is applied. Given a discrete distance measure $D : P \times P \to \mathbb{N}$ the distance distribution is defined as the relative frequency of the distance value $d \in \mathbb{N}$:

$$f(d) = \frac{2}{M(M-1)} |\{(s_i, s_j)|s_i, s_j \in P, i < j, D(s_i, s_j) = d\}|$$

The DDC is then defined as the Shannon entropy of the distribution of distance values:

$$DDC(P) := -\sum_d f(d) \log(f(d))$$

For the following analysis D is the Hamming-distance. This simple and fast measure is sufficient, because no shift, insert or delete operation is taking place. The DCC does not take into account a topology.

6.2 Visualisation of the Topology

In order to visualize the topology we map the state of the soup on two graphs. The first graph $G_1 = (V, E_1)$ should capture the connections imposed through the selection of b_2 by the function $chooseBucket_2(\ldots)$. It is defined as:

$$\begin{aligned} V &= \{0, \ldots, n_B - 1\}, \\ E_1 &= \{(b_1, b_2)|(b_1, b_2) \in V^2 \text{ and } \exists s_1 \in Bucket(b_1) : \\ &\qquad chooseBucket_1(\ldots, s_1) = b_2\} \end{aligned}$$

and its edges are displayed as dotted lines.

The edges for $G_2 = (V, E_2)$ are defined accordingly:

$$\begin{aligned} V &= \{0, \ldots, n_B - 1\}, \\ E_2 &= \{(b_2, b_3)|(b_2, b_3) \in V^2 \text{ and } \exists s_2 \in Bucket(b_2) : \\ &\qquad chooseBucket_2(\ldots, s_2) = b_3\} \end{aligned}$$

and are printed as solid lines.

6.3 Spatial Complexity

Spatial complexity can be measured by using another Shannon entropy based observable:
Let p_i be the probability, that a bucket i is selected to be the target for s_3 in one iteration of the hash-reactor algorithm. For a randomly initialized soup and a well defined hash function, p_i should be approximately $1/n_B$ for all $i \in \{0, \ldots, n_B - 1\}$. The spatial complexity is then defined as

$$SPC(P) = -\sum_{i=1}^{n_B - 1} p_i \log p_i. \qquad (2)$$

Here, p_i is estimated by running the algorithm for 10000 iterations without changing the state of the soup (skipping step 8) and counting how often the bucket i would be a target for the insertion of s_3.

7 Experiments

First we shall discuss a reference experiment.

7.1 A Well-stirred Tank Reactor

The hash-reactor algorithm is able model a well-stirred tank reactor by defining $chooseBucket_i(\ldots)$ to return a random bucket number. In such a case, no topological structure exists and every string reacts with any other string with equal probability. For a large reactor size the behavior of this system can be modeled — in principle — by of ordinary differential equations [17]. For the systems discussed here, however, the number of different string types is too high to make this method feasible.

Figure 3 shows an example of the system behavior for the well-stirred case. The time development of the concentration of some representative string types and total number of different string types is shown. During the first 20 generations a more or less complex evolutionary process can be observed, where new strings appear and "weaker" ones are replaced. After generation 20 this "pre-biotic" evolution declines and finally comes to an end. The remaining organization structure consists of very few (4) string types which are able to reproduce themselves. They are not able to generate totally new string types and innovativity is 0.

In this well-stirred reactor, the fastest replicating organization has superseded every other organization. As has been noted elsewhere, stable co-existence of many different species is practically impossible in well-stirred systems [8].

In the following subsections we shall describe two experiments making use of two different variants the hash-reactor algorithm and compare their behavior to the well-stirred case.

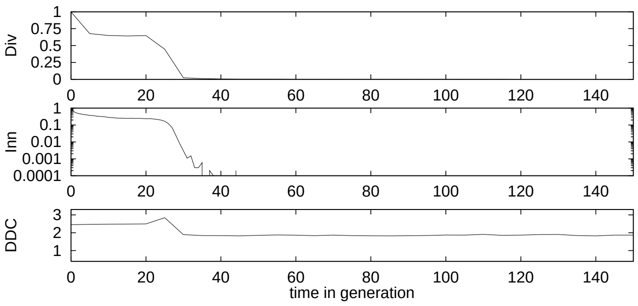

Figure 3: *Well-stirred tank reactor. $M = 10000$, automata reaction.*

7.2 Experiment 1

In this experiment the operator string s_1 and the operand string s_2 are selected randomly from the soup. The bucket where the product s_3 of their collision is stored is determined by a hash function that maps the product string s_3 to a distinct memory location. The key of s_3 is the string itself. The setup of the experiment reads:

$$\begin{aligned}
&chooseBucket_1() && = randomBucket() \\
&chooseBucket_2(\ldots) && = randomBucket() \\
&chooseBucket_3(\ldots, s_3) && = hash(key(s_3)) \\
&key(s) && = s, \quad s \in \{0,1\}^{32} \\
&hash(s) && = s \mod n_B \\
&n_B && = 10^4 \\
&n_{bs} && = 1 \\
&collision && = \text{automata reaction}
\end{aligned}$$

Note that the function $chooseBucket3(\ldots, s_3)$ is deterministic here. The soup is initialised with $M = n_B * n_{bs}$ random strings.

Typical resulting behaviour is shown in Figure 4. Compared to Figure 3 the diversity, complexity and innovativity are much higher in the long run. The soup is not overrun by a single (quasi)species. Many different organizations are coexisting, even if the simulation time is extended to more than 10,000 generations.

The innovativity is high because (i) the diversity is high *and* because (ii) every string can interact with every other string. With respect to the interaction, the soup is still well-stirred and the interaction between different string types is not constrained by the topology.

7.3 Experiment 2

Like in the previous experiment the location of the operator string s_1 is chosen randomly. But now the bucket the operand s_2 is chosen from is in turn determined by the structure of the operator s_1. The target bucket where the reaction product s_3 is stored is determined by the structure of s_2. In addition, we apply two different key functions to s_1 and s_2 respectively. $key_2(s_1)$ extracts 16 bits out of the center of s_1 and $key_3(s_2)$ extracts the rightmost 16 bits of s_2. The setup reads:

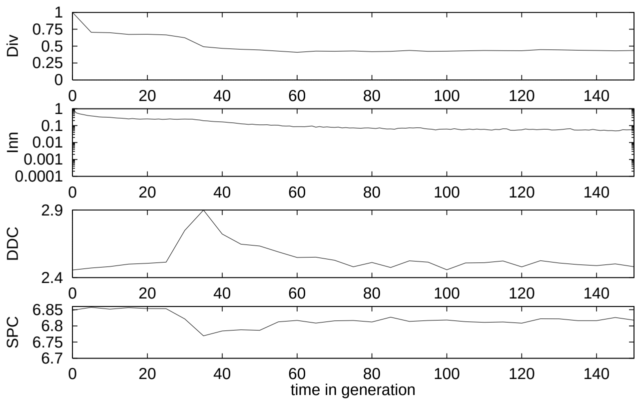

Figure 4: *Experiment 1. The diversity, innovativity, distance distribution complexity and spatial complexity are shown. $n_B = 1000, n_{bs} = 10$, soup initialized with 10000 random strings.*

$$\begin{aligned}
&chooseBucket_1() && = randomBucket() \\
&chooseBucket_2(\ldots, s_2) && = hash(key_2(s_1)) \\
&chooseBucket_3(\ldots, s_3) && = hash(key_3(s_2)) \\
&key_2(s) && = (s^{(23)}, \ldots, s^{(8)}) \\
&key_3(s) && = (s^{(15)}, \ldots, s^{(0)}) \\
&hash(k) && = \text{maximum entropy} \\
&n_B && = 10^2 \\
&n_{bs} && = 10 \\
&collision && = \text{automata reaction}
\end{aligned}$$

with $s = (s^{(31)}, s^{(30)}, \ldots, s^{(0)}) \in \{0,1\}^{32}$ and $k \in \{0,1\}^{16}$. In order to map the 16 bits of the key to a bucket number we have applied a hash function with maximum entropy. This means that there is no correlation between the key and the memory location it is mapped to. This is implemented by a deterministic pseudo-random number generator with high-dimensional internal state.

In Figure 5 the same macroscpic measurements are displayed as for the well-stirred case and Experiment 1. As expected, the diversity and innovativity is lower than in Experiment 1, but there are still many different string types.

Figure 6 shows a small time window of the soup. Three different dynamical behaviors can be observed. 1. Stasis: Bucket 122 (left of 123, unnumbered) shows no alterations. This is due to the fact that there is no reaction producing a product which is inserted into this bucket. 2. High activity: Bucket 123 shows a high turn-around of strings. The situation, however, is rather stable, because it can hardly be invaded by other string types. New string types are overwritten quickly. 3. Punctuated equilibrium: In bucket 125 a long periop of stasis is suddenly interruped by fluctuations leading to a new ensemble of strings.

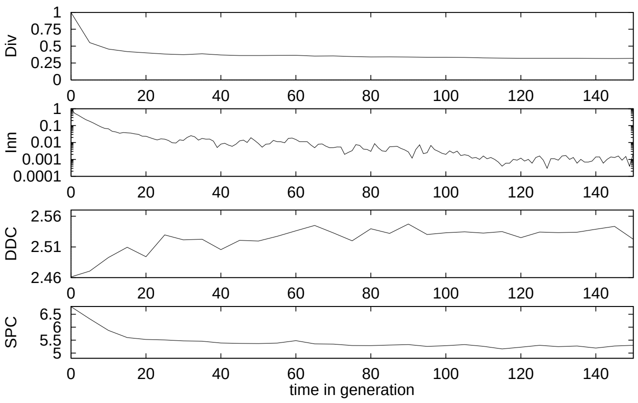

Figure 5: *Experiment 2. The diversity, innovativity, distance distribution complexity and spatial complexity are shown.* $n_B = 1000, n_{bs} = 10$*, soup initialized with 10000 random strings.*

The development of the topology in the vicinity of bucket 124 is depicted in Figure 7 and 8. The relationships to other buckets are reduced.

8 Discussion and Conclusion

- This contribution shows how to use efficient techniques of computer science to create complex self-organizing systems. We used hashing to create a topology which co-evolves with the objects (strings) in the system.
- The complexity of the behavior is high if one considers the simplicity of the algorithm and the small size of the objects (32 bit). The monitoring methods used here have shown some interesting aspects like the formation of a topology. For a deeper understanding, however, more analysis tools are needed.
- The system shows features like the formation of self-replicating autocatalytic networks, punctuated equilibria, development etc. It should be noted that we have not used any explicit variation operators like mutation (external noise) nor any explicit fitness function. Instead, variation is carried out by the strings themselves in their machine form. Therefore we have called this phenomenon **self-evolution**.

  The term "mutation" is used to characterize undirected variation in a genome. But here, the machines can "decide" how to change an operand. So, we suggest the term **active variation** partially following Ikegami and Hashimoto who call the variation in their system "active mutation" [10].
- Reaction systems – like the system discussed here – are able to perform useful computations [4]. Future research will aim at implications of artificial topological structures on the ability of computation.

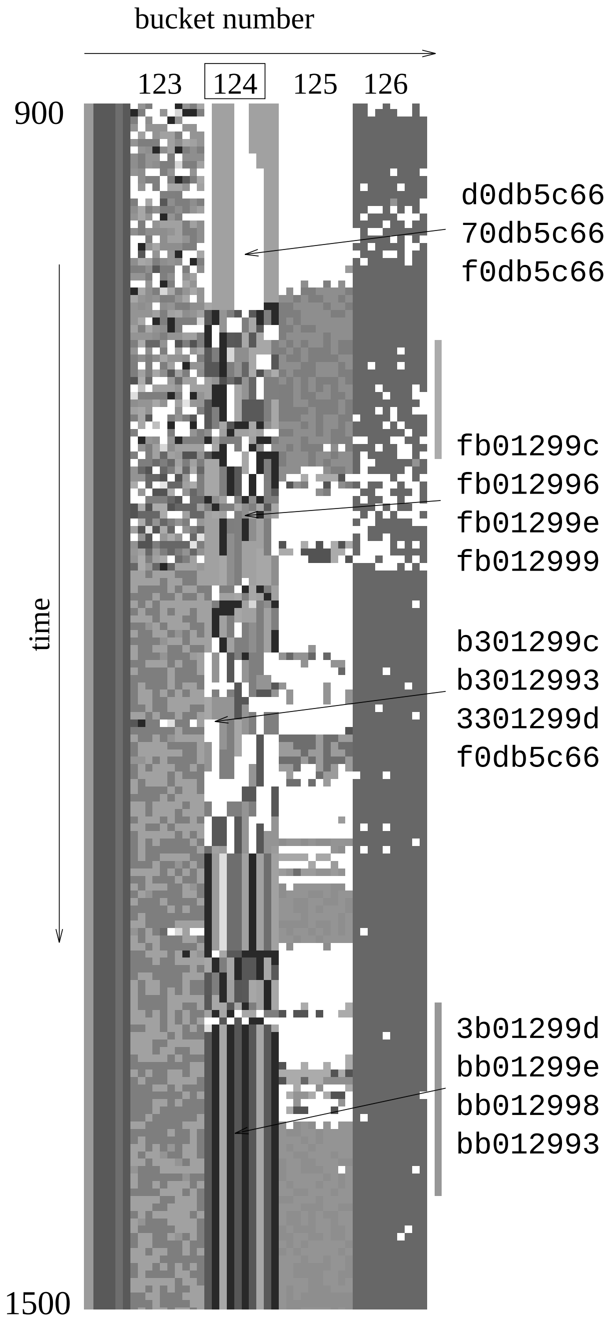

Figure 6: *Experiment 2: Buckets 122 (unnumbered) and 123-126 are shown for every 5th generation during the time 900 - 1500. Different colors are assigned randomly to string types. One pixel represents a single string in the soup.*

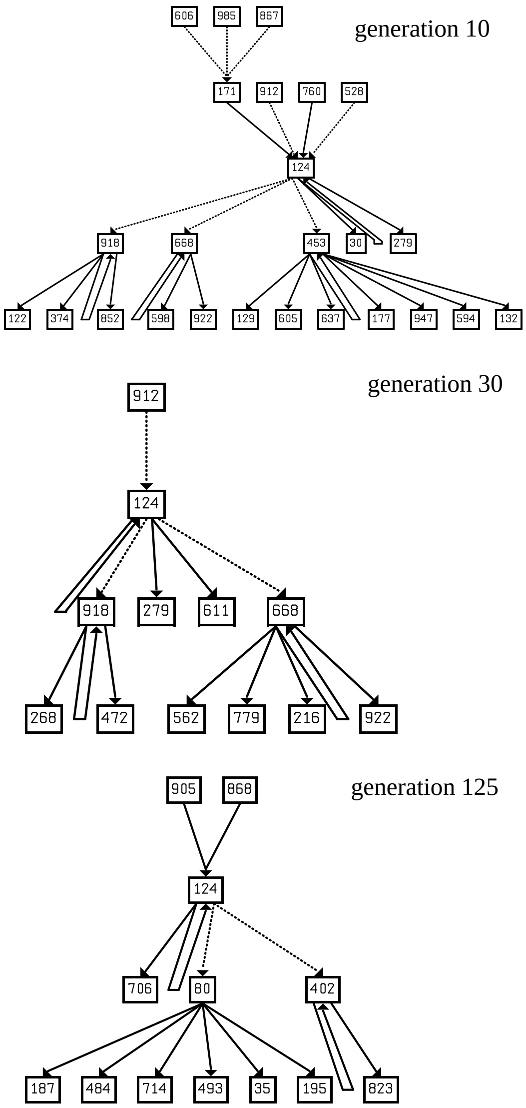

Figure 7: *Experiment 2: The early development of the topology in experiment 2 is shown in the vicinity of bucket 124 for generations 10, 30 and 125. The dotted edges represent the selection of the operand s_2 by the operator s_1 ($b_2 \leftarrow hash(key_2(s_1))$), the solid lines the selection of the target bucket b_3 by s_2 ($b_3 \leftarrow hash(key_3(s_2))$) respectively.*

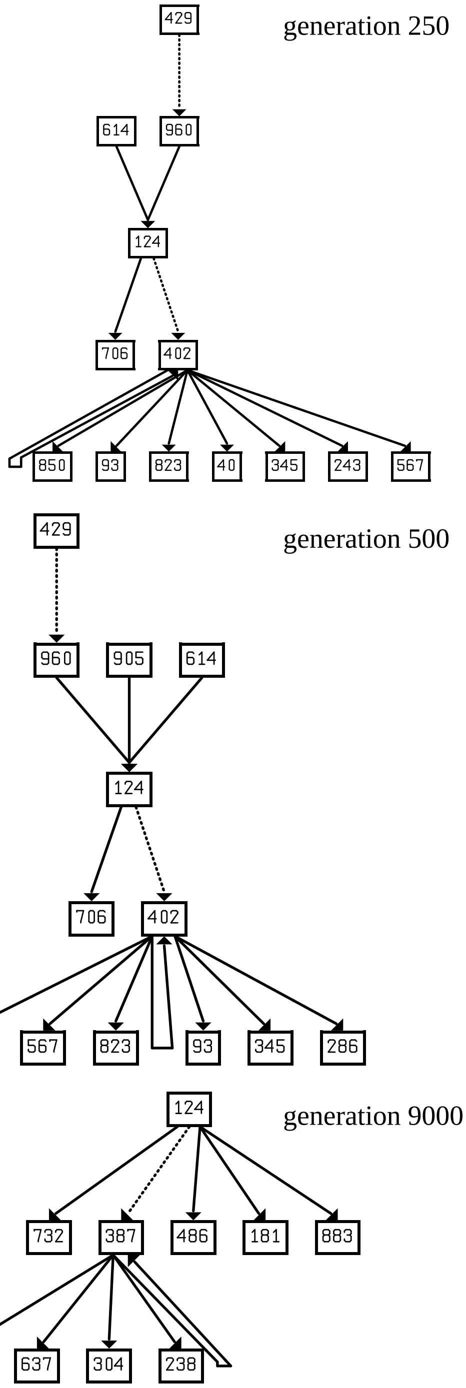

Figure 8: *Experiment 2: The topology in experiment 2 is shown in the vicinity of bucket 124 for generations 250, 500, 9000. The dotted edges represent the selection of the operand s_2 by the operator s_1 ($b_2 \leftarrow hash(key_2(s_1))$), the solid lines the selection of the target bucket b_3 by s_2 ($b_3 \leftarrow hash(key_3(s_2))$) respectively.*

Acknowledgments

This project is supported by the DFG (Deutsche Forschungsgemeinschaft) under grant Ba 1042/2-1.

References

[1] C. Adami. On modeling life. *Artificial Life*, 1:428–444, 1994.

[2] W. Banzhaf. Self-organizing algorithms derived from rna interactions. In Wolfgang Banzhaf and Frank H. Eeckman, editors, *Evolution and Biocomputing*, Volume 899 of *Lecture Notes in Computer Science*, pages 69–103. Springer, Berlin, 1995.

[3] W. Banzhaf. Self-replicating sequences of binary numbers. *Computers and Mathematics* (26), 1–8. 1993

[4] W. Banzhaf, P. Dittrich and H. Rauhe. Emergent computation by catalytic reactions. *Nanotechnology*, 7: 307–314, 1996.

[5] M. A. Bedau. Three illustrations of artificial life's working hypothesis. In Wolfgang Banzhaf and Frank H. Eeckman, editors, *Evolution and Biocomputing*, Volume 899 of *Lecture Notes in Computer Science*, pages 53–68. Springer, Berlin, 1995.

[6] E. Berlekamp, J. Conway, and R. Guy. *Winning Ways for your mathematical plays, Vol. II.* Academic Press, New York, 1982

[7] P. Dittrich. ANSI C source code for the automata reactions. Internetserver of the Chairt of Systems Analysis, Dept. of Computer Science, University of Dortmund [3], 1997.

[8] W. Fontana. Algorithmic chemistry. In Christopher G. Langton, Charles Taylor, J. Doyne Farmer, and Steen Rasmussen, editors, *Artificial Life II: proceedings of the second artificial life workshop*, pages 159–209. Addison-Wesley, Redwood City, CA, 1992.

[9] D. R. Hofstadter. *Goedel, Escher, Bach: An Eternal Golden Braid.* Basic Books,Inc., New York, 1979.

[10] T. Ikegami and T. Hashimoto. Coevolution of machines and tapes. In F. Morán, A. Moreno, J. J. Merelo, and P. Chacón, editors, *Proceedings of the Third European Conference on Artificial Life : Advances in Artificial Life*, volume 929 of *LNAI*, pages 234–245. Springer Verlag, Berlin, 1995.

[3] file://ls11-www.informatik.uni-dortmund.de/pub/biocomp/src/

[11] S. A. Kauffman. *The Origins of Order.* Oxford University Press, 1993.

[12] J. T. Kim. Using distance distribution to measure complexity of populations. In D.J. Stein and L. Nadel, editors, *1993 Lectures in Complex Systems.* Addison-Wesley, Redwood City, CA, 1995.

[13] C. G. Langton. Artificial life. In Christopher G. Langton, editor, *Artificial Life*, SFI Studies in the Science of Complexity, pages 1–45. Santa Fe Institute, Los Alamos, New Mexico, Addison-Wesley, Redwood City, CA, 1989.

[14] A. N. Pargellis. The spontanous generation of digital "life". *Physica D*, 91:86–96, 1996. AL1.

[15] A. S. Perelson and S. A. Kauffman, editors. *Molecular Evolution on Rugged Landscapes: Proteins, RNA and the Immune System*, SFI Studies in the Science of Complexity, vol. IX. Addison-Wesley, Redwood City, California, 1991.

[16] T. S. Ray. An approach to the synthesis of life. In Christopher G. Langton, Charls Taylor, J. Doyne Farmer, and Steen Rasmussen, editors, *Artificial Life II: proceedings of the second artificial life workshop*, pages 371–408. Addison-Wesley, Redwood City, CA, 1992.

[17] P. F. Stadler, W. Fontana, and J. H. Miller. Random catalytic reaction networks. *Physica D*, 63:378–392, November 1993.

[18] M. Thürk. *Ein Modell zur Selbstorganisation von Automatenalgorithmen zum Studium molekularer Evolution.* PhD thesis, University of Jena, naturwissenschaftliche Fakultẗ, 1993.